(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 426 410 B1

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:
19.06.1996 Bulletin 1996/25

(51) Int Cl.6: **G06F 1/32**

(21) Application number: **90311832.1**

(22) Date of filing: **29.10.1990**

(54) Real-time power conservation for portable computers

Echtzeitleistungseinsparung für tragbare Rechner

Conservation d'énergie en temps réel pour ordinateurs portables

(84) Designated Contracting States:
DE FR GB IT NL

(30) Priority: **30.10.1989 US 429270**

(43) Date of publication of application:
08.05.1991 Bulletin 1991/19

(73) Proprietor: **TEXAS INSTRUMENTS INCORPORATED**
Dallas Texas 75265 (US)

(72) Inventors:
- **Watts, Lavaughn F.,Jr.**
  Temple, Texas (US)
- **Wallace, Steven J.**
  Temple, Texas (US)

(74) Representative: **Abbott, David John et al**
Abel & Imray
Northumberland House
303-306 High Holborn
London, WC1V 7LH (GB)

(56) References cited:
EP-A- 0 363 567
WO-A-86/00432
US-A- 4 686 386

- **IBM Technical Disclosure Bulletin, vol. 29, no. 9, Febr. 1987, Armonk, New York, US, pages 4122-4124, "System power savings by automatic sleep mode"**

EP 0 426 410 B1

Printed by Jouve, 75001 PARIS (FR)

Description

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to real-time computer power conservation, and more particularly to an apparatus and method for reduction of central processing unit (CPU) clock time based on the real-time activity level within the CPU of a portable computer.

Description of the Related Art

The International Patent Application published under the number WO A 86 00432 discloses a computer having a variable frequency clock. The computer maintains a table recording the number of jobs to be processed categorised according to the number of cycles that they take to execute. As jobs are requested and completed the table is adjusted. Periodically a figure for the desired operating frequency of the CPU is calculated from the numbers in the table and weighting figures for each category. The frequency of the clock is then set accordingly.

The IBM Technical Disclosure Bulletin, vol.29, no. 9, pages 4122 to 4124 discloses a computer having a CPU having a sleep mode in which the clock rate is minimal. When the computer has been waiting for more than a certain period for a key on the keyboard to be pressed by the user it switches itself into the sleep mode. Full processing speed is restored on interrupt from the keyboard when a key is pressed.

The European patent application published under the number EP 0 363 567 A2 or 18.04.90 discloses a computer system having a variable frequency clock. On interrupt an interrupt handler module of the operating system selects a appropriate speed for the clock, for example, to keep the processor in synchronism with an I/O adapter card. When the interrupt has been processed the previous clock speed is restored.

During the development stages of personal computers, the transportable or portable computer has become very popular. Such portable computer uses a large power supply and really represents a small desktop personal computer. Portable computers are smaller and lighter than a desktop personal computer and allow an user to employ the same software that can be used on a desktop computer.

The first generation "portable" computers only operated from an A/C wall power. As personal computer development continued, battery-powered computers were designed. Furthermore, real portability became possible with the development of new display technology, better disk storage, and lighter components.

However, the software developed was desiged to run on a desk top personal computers, with all the features of a computer, without regard to battery-powered portable computers that only had limited amounts of power available for short periods of time. No special considerations were made by the software, operating system (MS-DOS), Basic Input/Output System (BIOS), or the third party application software to conserve power usage for these portable computers.

As more and more highly functional software packages were developed, desk top computer users experienced increased performance from the introductions of higher computational CPUs, increased memory, and faster high performance disk drives.

Unfortunately, portable computers continued to run only on A/C power or with large and heavy batteries. In trying to keep up with the performance requirements of the desk top computers, and the new software, expensive components were used to cut the power requirements. Even so, the heavy batteries still did not run very long. This meant users of portable computers had to settle for A/C operation or very short battery operation to have the performance that was expected from the third party software.

Portable computer designers stepped the performance down to 8088- and 8086-type processors to reduce the power consumption. The supporting circuits and CPU took less power to run and therefore, lighter batteries could be used. Unfortunately, the new software requiring 80286-type instructions, that did not exist in the older slower 8088/8086 CPUs, did not run.

In an attempt to design a portable computer that could conserve power, thereby yielding longer battery operation, smaller units, and less weight, some portable computer designers proceeded to reduce power consumption of a portable computer while an user is not using the computer. For example, designers obtain a reduction in power usage by slowing or stopping the disk drive after some predetermined period of inactivity; if the disk drive is not being used, the disk drive is turned off, or simply placed into a standby mode. When the user is ready to use the disk, the operator must wait until the disk drive is spinned up and the computer system is ready again for full performance before the operator may proceed with the operation.

Other portable computer designers conserve power by turning the computer display off when the keyboard is not being used. However, in normal operation the computer is using full power. In other words, power conservation by this

method is practical only when the user is not using the components of the system. It is very likely, however, that the user will turn the computer off when not in use.

Nevertheless, substantial power conservation while the operator is using the computer for meaningful work is needed. When the operator uses the computer, full operation of all components is required. During the intervals while the operator is not using the computer, however, the computer could be turned off or slowed down to conserve power consumption. It is critical to maintaining performance to determine when to slow the computer down or turn it off without disrupting the user's work, upsetting the third party software, or confusing the operating system, until operation is needed.

Furthermore, although an user can wait for the disk to spin up as described above, application software packages cannot wait for the CPU to "spin up" and get ready. The CPU must be ready when the application program needs to compute. Switching to full operation must be completed quickly and without the application program being affected. This immediate transition must be transparent to the user as well as to the application currently active. Delays cause user operational problems in response time and software compatability, as well as general failure by the computer to accurately execute a required program.

Other attempts at power conservation for portable computers include providing a "Shut Down" or "Standby Mode" of operation. The problem, again, is that the computer is not usable by the operator during this period. The operator could just as well turned off the power switch of the unit to save power. This type of power conservation only allows the portable computer to "shut down" and thereby save power if the operator forgets to turn off the power switch, or walks away from the computer for the programmed length of time. The advantage of this type of power conservation over just turning the power switch off/on is a much quicker return to full operation. However, this method of power conservation is still not real-time, intelligent power conservation while the computer is on and processing data which does not disturb the operating system, BIOS, and any third party application programs currently running on the computer.

Some attempt to meet this need was made by VLSI vendors in providing circuits that either turned off the clocks to the CPU when the user was not typing on the keyboard or woke up the computer on demand when a keystroke occurred. Either of these approaches reduce power but the computer is dead (unusable) during this period. Background operations such as updating the system clock, communications, print spooling, and other like operations cannnot be performed. Some existing portable computers employ these circuits. After a programmed period of no activity, the computer turns itself off. The operator must turn the machine on again but does not have to reboot the operating system and application program. The advantage of this circuity is, like the existing "shut down" operations, a quick return to full operation without restarting the computer. Nevertheless, this method only reduces power consumption when the user walks away from the machine and does not actually extend the operational life of the battery charge.

SUMMARY OF THE INVENTION

In view of the above problems associated with the related art, it is an object of the present invention to provide an apparatus and method for real-time conservation of power for computer systems without any real-time performance degradation, such conservation of power remaining transparent to the user.

Another object of the present invention is to provide an apparatus and method for predicting the activity level within a computer system and using the prediction for automatic power conservation.

Yet another object of the present invention is to provide an apparatus and method which allows user modification of automatic activity level predictions and using the modified predictions for automatic power conservation.

A further object of the present invention is to provide an apparatus and method for real-time reduction and restoration of clock speeds thereby returning the CPU to full processing rate from a period of inactivity which is transparent to software programs.

These objects are accomplished in a preferred embodiment of the present invention by an apparatus and method which determine whether a CPU may rest based upon the CPU activity level and activates a hardware selector based upon that determination. If the CPU may rest, or sleep, the hardware selector applies oscillations at a sleep clock level; if the CPU is to be active, the hardware selector applies oscillations at a high speed clock level.

According to a first aspect of the invention there is provided a method for implementing a real-time power conservation mode in a computer having a central processing unit (CPU), comprising the sequential steps of:

determining in real time the level of CPU activity by detecting periods of activity and inactivity of said CPU;
deciding from said level of CPU activity whether it is available for power conservation;
if said CPU is available for power conservation, causing a hardware selector to reduce the clock rate below the current clock rate or stop the clock provided to said CPU;
maintaining the clock stopped or at the reduced rate until an interrupt occurs, and
causing said hardware selector to restore the clock rate provided to said CPU to said current clock rate in response

to said interrupt.

According to a second aspect of the invention there is provided an apparatus for real time power conservation in a computer having a central processing unit (CPU), comprising :

a first clock oscillator for providing pulses at a first rate;
a second clock oscillator for providing pulses at a second rate, slower than said first rate; and
a hardware selector for selecting pulses from said first oscillator or from said second oscillator, said hardware selector being arranged to apply the selected pulses as clock signals to said CPU; the apparatus being characterized by further comprising :
a CPU activity detector for determining in real time the level of CPU activity by detecting periods of activity and inactivity of said CPU and providing an indication of that level of CPU activity; and
a CPU sleep manager adapted to receive the indication of the level of CPU activity from said CPU activity detector, and apply a signal to said hardware selector designating which pulses said hardware selector should select to apply clock signals to said CPU, according to said received indication.

The present invention examines the state of CPU activity, as well as the activity of both the operator and any application software program currently active. This sampling of activity is performed real-time, adjusting the performance level of the computer to manage power conservation and computer power. These adjustments are accomplished within the CPU cycles and do not affect the user's perception of performance.

Thus, when the operator for the third party software of the operating system/BIOS is not using the computer, the present invention will effect a quick turn off or slow down of the CPU until needed, thereby reducing the power consumption, and will promptly restore full CPU operation when needed without affecting perceived performance. This switching back into full operation from the "slow down" mode occurs without the user having to request it and without any delay in the operation of the computer while waiting for the computer to return to a "ready" state.

These and other features and advantages of the invention will be apparent to those skilled in the art from the following detailed description of a preferred embodiment, taken together with the accompanying drawings, in which:

DESCRIPTION OF THE DRAWINGS:

FIG. 1 is a flowchart depicting the self-tuning aspect of a preferred embodiment of the present invention;
FIGs. 2a-2d are flowcharts depicting the active power conservation monitor employed by the present invention;
FIG. 3 is a simplified schematic diagram representing the active power conservation associated hardware employed by the present invention;
FIG. 4 is a schematic of the sleep hardware for one embodiment of the present invention; and
FIG. 5 is a schematic of the sleep hardware for another embodiment of the present invention.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

If the period of computer activity in any given system is examined, the CPU and associated components have a utilization percentage. If the user is inputing data from the keyboard, the time between keystrokes is very long in terms of CPU cycles. Many things can be accomplished by the computer during this time, such as printing a report. Even during the printing of a report, time is still available for additional operations such as background updating of a clock/calendar display. Even so, there is almost always spare time when the CPU is not being used. If the computer is turned off or slowed down during this spare time, then power consumption is obtained real-time. Such real-time power conservation extends battery operation life.

According to the preferred embodiment of the present invention, to conserve power under MS-DOS, as well as other operating systems such as OS/2, XENIX, and those for Apple computers, requires a combination of hardware and software. It should be noted that because the present invention will work in any system, while the implementation may vary slightly on a system-by-system basis, the scope of the present invention should therefore not be limited to computer systems operating under MS/DOS.

Slowing down or stopping the computer system components according to the preferred embodiment of the present invention, reduces power consumption, although the amount of power saved may vary. Therefore, according to the present invention, stopping the clock (where possible as some CPUs cannot have their clocks stopped) reduces the power consumption more than just slowing the clock.

In general, the number of operations (or instructions) per second may be considered to be roughly proportional to the processor clock:

$$\text{instructions/second} = \text{instructions/cycle} * \text{cycles/second}$$

Assuming for simplicity that the same instruction is repeatedly executed so that instructions/second is constant, the relationship can be expressed as follows:

$$Fq = K_1 * Clk$$

where Fq is instructions/second, K_1 is constant equal to the instructions/cycle, and Clk equals cycles/second. Thus, roughly speaking, the rate of execution increases with the frequency of the CPU clock.

The amount of power being used at any given moment is also related to the frequency of the CPU clock and therefore to the rate of execution. In general this relationship can be expressed as follows:

$$P = K_2 + (K_3 * Clk)$$

where P is power in watts, K_2 is a constant in watts, K_3 is a constant and expresses the number of watt-seconds/cycle, and Clk equals the cycles/second of the CPU clock. Thus it can also be said that the amount of power being consumed at any given time increases as the CPU clock frequency increases.

Assume that a given time period T is divded into N intervals such that the power P was constant during each interval. Then the amount of energy E expended during T would be given by:

$$E = P(1)\text{delta}T_1 + P(2)\text{delta}T_2 \ldots + P(N)\text{delta}T_N$$

Further assume that the CPU clock "Clk" has only two states, either "ON" or "OFF". For the purposes of this discussion, the "ON" state represents the CPU clock at its maximum frequency, while the "OFF" state represents the minimum clock rate at which the CPU can operate (this may be zero for CPUs that can have their clocks stopped). For the condition in which the CPU clock is always "ON", each P(i) in the previous equation is equal and the total energy is:

$$E(max) = P(on) * (\text{delta } T_1 + \text{delta } T_2 \ldots + \text{delta } T_N)$$

$$= P(on) * T$$

This represents the maximum power consumption of the computer in which no power conservation measures are being used. If the CPU clock is "off" during a portion of the intervals, then there are two power levels possible for each interval. The P(on) represents the power being consumed when the clock in in its "ON" state, while P(off) represents the power being used when the clock is "OFF". If all of the time intervals in which the clock is "ON" is summed into the quantity "T(on)" and the "OFF" intervals are summed into "T(off)", then it follows:

$$T = T(on) + T(off)$$

Now the energy being used during period T can be written:

$$E = [P(on) * T(on)] + [P(off) * T(off)]$$

Under these conditions, the total energy consumed may be reduced by increasing the time intervals T(off). Thus, by controlling the periods of time the clock is in its "OFF" state, the amount of energy being used may be reduced. If the T(off) period is divided into a large number of intervals during the period T, then as the width of each interval goes to zero, energy consumption is at a maximum. Conversely, as the width of the T(off) intervals increase, the energy consumed decreases.

If the "OFF" intervals are arranged to coincide with periods during which the CPU is normally inactive, then the user cannnot perceive any reduction in performance and overall energy consumption is reduced from the E(max) state. In order to align the T(off) intervals with periods of CPU inactivity, the CPU activity level is used to determine the width of the T(off) intervals in a closed loop. Figure 1 depicts such a closed loop. The activity level of the CPU is determined at Step 10. If this level is an increase over an immediately previous determination, the present invention decreases the T(off) interval (Step 20) and returns to determine the activity level of the CPU again. If, on the other hand, this activity level is a decrease over an immediately previous determination, the present invention increases the T(off) interval (Step 30) and proceeds to again determine the activity level of the CPU. Thus the T(off) intervals are constantly being adjusted to match the system activity level.

In any operating system, two key logic points exist: an IDLE, or "do nothing", loop within the operating system and an operating system request channel, usually available for services needed by the application software. By placing logic inline with these logic points, the type of activity request made by an application software can be evaluated, power conservation can be activated and slice periods determined. A slice period is the number of T(on) vs. T(off) intervals over time, computed by the activity level. An assumption may be made to determine CPU activity level: Software programs that need service usually need additional services and the period of time between service requests can be used to determine the activity level of any application software running on the computer and to provide slice counts

for power conservation according to the present invention.

Once the CPU is interrupted during a power conservation slice (T(off)), the CPU will save the interrupted routine's state prior to vectoring to the interrupt software. Of course, since the power conservation software was operating during this slice, control will be returned to the active power conservation loop (monitor 40) which simply monitors the CPU's clock to determine an exit condition for the power conservation mode, thereby exiting from T(off) to T(on) state. The interval of the next power conservation state is adjusted by the activity level monitored, as discussed above in connection with Figure 1. Some implementations can create an automatic exit from T(off) by the hardware logic, thereby forcing the power conservation loop to be exited automatically and executing an interval T(on).

More specifically, looking now at Figures 2a-2d, which depict the active power conservation monitor 40 of the present invention. The CPU installs monitor 40 either via a program stored in the CPU ROM or loads it from an external device storing the program in RAM. Once the CPU has loaded monitor 40, it continues to INIT 50 for system interrupt initialization, user configurational setup, and system/application specific initialization. IDLE branch 60 (more specifically set out in Figure 2b) is executed by a hardware or software interrupt for an IDLE or "do nothing" function. This type of interrupt is caused by the CPU entering either an IDLE or a "do nothing" loop (i.e., planned inactivity). The ACTIVITY branch 70 of the flowchart, more fully described below in relation to Figure 2d, is executed by a software or hardware interrupt due to an operating system or I/O service request, by an application program or internal operating system function. An I/O service request made by a program may, for example, be a disk I/O, read, print, load, etc. Regardless of the branch selected, control is eventually returned to the CPU operating system at RETURN 80. The INIT branch 50 of this flowchart, shown in Figure 2a, is executed only once if it is loaded via program into ROM or is executed every time during power up if it is loaded from an external device and stored in the RAM. Once this branch of active power monitor 40 has been fully executed, whenever control is yielded from the operating system to the power conservation mode, either IDLE 60 or ACTIVITY 70 branches are selected depending on the type of CPU activity: IDLE branch 60 for power conservation during planned inactivity and ACTIVITY branch 70 for power conservation during CPU activity.

Looking more closely at INIT branch 50 , after all system interrupt and variables are initialized, the routine continues at Step 90 to set the Power_level equal to DEFAULT_LEVEL. In operating systems where the user has input control for the Power_level, the program at Step 100 checks to see if a User_level has been selected. If the User_level is less than zero or greater than the MAXIMUM_LEVEL, the system uses the DEFAULT_LEVEL. Otherwise, it continues onto Step 110 where it modifies the Power_level to equal the User_level.

According to the preferred embodiment of the present invention, the system at Step 120 sets the variable Idle_tick to zero and the variable Activity_tick to zero. Under an MS/DOS implementation, Idle_tick refers to the number of interrupts found in a "do nothing" loop. Activity_tick refers to the number of interrupts caused by an activity interrupt which in turn determines the CPU activity level. Tick count represents a delta time for the next interrupt. Idle_tick is a constant delta time from one tick to another (interrupt) unless overwritten by a software interrupt. A software interrupt may reprogram delta time between interrupts.

After setting the variables to zero, the routine continues on to Setup 130 at which time any application specific configuration fine-tuning is handled in terms of system-specific details and the system is initialized. Next the routine arms the interrupt I/O (Step 140) with instructions to the hardware indicating the hardware can take control at the next interrupt. INIT branch 50 then exits to the operating system, or whatever called the active power monitor originally, at RETURN 80.

Consider now IDLE branch 60 of active power monitor 40, more fully described at Figure 2b. In response to a planned inactivity of the CPU, monitor 40 (not specifically shown in this Figure) checks to see if entry into IDLE branch 60 is permitted by first determining whether the activity interrupt is currently busy. If Busy_A equals BUSY_FLAG (Step 150), which is a reentry flag, the CPU is busy and cannot now be put to sleep. Therefore, monitor 40 immediately proceeds to RETURN I 160 and exits the routine. RETURN I 160 is an indirect vector to the previous operating system IDLE vector interrupt for normal processing stored before entering monitor 40. (I.e., this causes an interrupt return to the last chained vector.)

If the Busy_A interrupt flag is not busy, then monitor 40 checks to see if the Busy_Idle interrupt flag, Busy_I, equals BUSY_FLAG (Step 170). If so, this indicates the system is already in IDLE branch 60 of monitor 40 and therefore the system should not interrupt itself. If Busy_I = BUSY_FLAG, the system exits the routine at RETURN I indirect vector 160.

If, however, neither the Busy_A reentry flag or the Busy_I reentry flag have been set, the routine sets the Busy_I flag at Step 180 for reentry protection (Busy_I = BUSY_FLAG). At Step 190 Idle_tick is incremented by one. Idle_tick is the number of T(on) before a T(off) interval and is determined from IDLE interrupts, setup interrupts and from CPU activity level. Idle_tick increments by one to allow for smoothing of events, thereby letting a critical I/O activity control smoothing.

At Step 200 monitor 40 checks to see if Idle_tick equals IDLE_MAXTICKS. IDLE_MAXTICKS is one of the constants initialized in Setup 130 of INIT branch 50, remains constant for a system, and is responsible for self-tuning of the activity level. If Idle_tick does not equal IDLE_MAXTICKS, the Busy_I flag is cleared at Step 210 and exits the loop proceeding to the RETURN I indirect vector 160. If, however, Idle_tick equals IDLE_MAXTCKS, Idle_tick is set equal

to IDLE_START_TICKS (Step 220). IDLE_START_TICKS is a constant which may or may not be zero (dpending on whether the particular CPU can have its clock stopped). This step determines the self-tuning of how often the rest of the sleep functions may be performed. By setting IDLE_START_TICKS equal to IDLE_MAXTICKS minus one, a continuous T(off) interval is achieved. At Step 230, the Power_level is checked. If it is equal to zero, the monitor clears the Busy_I flag (Step 210), exits the routine at RETURN I 160, and returns control to the operating system so it may continue what it was originally doing before it entered active power monitor 40.

If, however, the Power_level does not equal zero at Step 240, the routine determines whether an interrupt mask is in place. An interrupt mask is set by the system/application software, and determines whether interrupts are available to monitor 40. If interrupts are NOT_AVAILABLE, the Busy_I reentry flag is cleared and control is returned to the operating system to continue what it was doing before it entered monitor 40. Operating systems, as well as application software, can set T(on) interval to yield a continuous T(on) state by setting the interrupt mask equal to NOT_AVAILABLE.

Assuming an interrupt is AVAILABLE, monitor 40 proceeds to the SAVE POWER subroutine 250 which is fully executed during one T(off) period established by the hardware state. (For example, in the preferred embodiment of the present invention, the longest possible interval could be 18 ms, which is the longest time between two ticks or interrupts from the real-time clock.) During the SAVE POWER subroutine 250, the CPU clock is stepped down to a sleep clock level.

Once a critical I/O operation forces the T(on) intervals, the IDLE branch 60 interrupt tends to remain ready for additional critical I/O requests. As the CPU becomes busy with critical I/O, less T(off) intervals are available. Conversely, as critical I/O requests decrease, and the time intervals between them increase, more T(off) intervals are available. IDLE branch 60 is a self-tuning system based on feedback from activity interrupts and tends to provide more T(off) intervals as the activity level slows. As soon as monitor 40 has completed SAVE POWER subroutine 250, shown in Figure 2c and more fully described below, the Busy_I reentry flag is cleared (Step 210) and control is returned at RETURN I 160 to whatever operating system originally requested monitor 40.

Consider now Figure 2c, which is a flowchart depicting the SAVE POWER subroutine 250. Monitor 40 determines what the I/O hardware high speed clock is at Step 260. It sets the CURRENT_CLOCK_RATE equal to the relevant high speed clock and saves this value to be used for CPUs with multiple level high speed clocks. Thus, if a particular CPU has 12 MHz and 6 MHz high speed clocks, monitor 40 must determine which high speed clock the CPU is at before monitor 40 reduces power so it may reestablish the CPU at the proper high speed clock when the CPU awakens. At Step 270, the Save_clock_rate is set equal to the CURRENT_CLOCK_RATE determined. Save_clock_rate 270 is not used when there is only one high speed clock for the CPU. Monitor 40 now continues to SLEEPCLOCK 280, where a pulse is sent to the hardware selector (shown in Figure 3) to put the CPU clock to sleep (i.e., lower or stop its clock frequency). The I/O port hardware sleep clock is at much lower oscillations than the CPU clock normally employed.

At this point either of two events can happen. A system/application interrupt may occur or a real-time clock interrupt may occur. If a system/application interrupt 290 occurs, monitor 40 proceeds to interrupt routine 300, processing the interrupt as soon as possible, arming interrupt I/O at Step 310, and returning to determine whether there has been an interrupt (Step 320). Since in this case there has been an interrupt, the Save_clock_rate is used (Step 330) to determine which high speed clock to return the CPU to and SAVE POWER subroutine 250 is exited at RETURN 340. If, however, a system/application interrupt is not received, the SAVE POWER subroutine 250 will continue to wait until a real-time clock interrupt has occurred (Step 320). Once such an interrupt has occurred, SAVE POWER subroutine 250 reestablishes the CPU at the stored Save-clock-rate. If the sleep clock rate was not stopped, in other words, the sleep clock rate was not zero, control is passed at a slow clock and SAVE POWER subroutine 250 will execute interrupt loop 320 several times. If however, control is passed when the sleep clock rate was zero, in other words, there was no clock, the SAVE POWER subroutine 250 will execute interrupt loop 320 once before returning the CPU clock to the Save_clock_rate 330 and exiting (Step (340).

Consider now Figure 2d which is a flowchart showing ACTIVITY branch 70 triggered by an application/system activity request via an operating system service request interrupt. ACTIVITY branch 70 begins with reentry protection. Monitor 40 determines at Step 350 whether Busy_I has been set to BUSY_FLAG. If it has, this means the system is already in IDLE branch 60 and cannot be interrupted. If Busy_I = BUSY_FLAG, monitor 40 exits to RETURN I 160, which is an indirect vector to an old activity vector interrupt for normal processing, via an interrupt vector after the operating system performs the requested service.

If however, the Busy_I flag does not equal BUSY_FLAG, which means IDLE branch 60 is not being accessed, monitor 40 determines at Step 360 if the BUSY_A flag has been set equal to BUSY-FLAG. If so, control will be returned to the system at this point because ACTIVITY branch 70 is already being used and cannot be interrupted. If the Busy_A flag has not been set, in other words, Busy_A does not equal BUSY_FLAG, monitor 40 sets Busy_A equal to BUSY_FLAG at Step 370 so as not to be interrupted during execution of ACTIVITY branch 70. At Step 380 the Power_level is determined. If Power_level equals zero, monitor 40 exits ACTIVITY branch 70 after clearing the Busy_A reentry flag (Step 390). If however, the Power_level does not equal zero, the CURRENT_CLOCK_RATE of the I/O hardware is next determined. As was true with Step 270 of Figure 2C, Step 400 of Figure 2d uses the

CURRENT_CLOCK_RATE if there are multiple level high speed clocks for a given CPU. Otherwise, CURRENT_CLOCK_RATE always equals the CPU high speed clock. After the CURRENT_CLOCK_RATE is determined (Step 400), at Step 410 Idle_tick is set equal to the constant START_TICKS established for the previously determined CURRENT_CLOCK_RATE. T(off) intervals are established based on the current high speed clock that is active.

Monitor 40 next determines that a request has been made. A request is an input by the application software running on the computer, for a particular type of service needed. At Step 420, monitor 40 determines whether the request is a CRITICAL I/O. If the request is a CRITICAL I/O, it will continuously force T(on) to lengthen until the T(on) is greater than the T(off), and monitor 40 will exit ACTIVITY branch 70 after clearing the Busy_A reentry flag (Step 390). If, on the other hand, the request is not a CRITICAL I/O, then the Activity_tick is incremented by one at Step 430. It is then determined at Step 440 whether the Activity_tick now equals ACTIVITY_ MAXTICKS. Step 440 allows a smoothing from a CRITICAL I/O, and makes the system ready from another CRITICAL I/O during Activity_tick T(on) intervals. Assuming Activity_tick does not equal ACTIVITY_MAXTICKS, ACTIVITY branch 70 is exited after clearing the Busy_A reentry flag (Step 390). If, on the other hand, the Activity_tick equals constant ACTIVITY_MAXTICKS, at Step 450 Activity_tick is set to the constant LEVEL_MAXTICKS established for the particular Power_level determined at Step 380.

Now monitor 40 determines whether an interrupt mask exists (Step 460). An interrupt mask is set by system/application software. Setting it to NOT_AVAILABLE creates a continuous T(on) state. If the interrupt mask equals NOT_AVAILABLE, there are no interrupts available at this time and monitor 40 exits ACTIVITY branch 70 after clearing the Busy_A reentry flag (Step 390). If, however, an interrupt is AVAILABLE, monitor 40 determines at Step 470 whether the request identified at Step 420 was for a SLOW I/O_INTERRUPT. SLOW I/O requests may have a delay until the I/O device becomes "ready". During the "make ready" operation, a continuous T(off) interval may be set up and executed to conserve power. Thus, if the request is not a SLOW I/O_INTERRUPT, ACTIVITY branch 70 is exited after clearing the Busy_A reentry flag (Step 390). If, however, the request is a SLOW I/O_INTERRUPT, and time yet exists before the I/O device becomes "ready", monitor 40 then determines at Step 480 whether the I/O request is COMPLETE (i.e., is I/O device ready?). If the I/O device is not ready, monitor 40 forces T(off) to lengthen, thereby forcing the CPU to wait, or sleep, until the SLOW I/O device is ready. At this point it has time to save power and ACTIVITY branch 70 enters SAVE POWER subroutine 250 previously described in connection with to Figure 2C. If, however, the I/O request is COMPLETE, control is returned to the operating system subsequently to monitor 40 exiting ACTIVITY branch 70 after clearing Busy_A reentry flag (Step 390).

Self-tuning is inherent within the control system of continuous feedback loops. The software of the present invention can detect when CPU activity is low and therefore when the power conservation aspect of the present invention may be activated. Once the power conservation monitor is activated, a prompt return to full speed CPU clock operation within the interval is achieved so as to not degrade the performance of the computer. To achieve this prompt return to full speed CPU clock operation, the preferred embodiment of the present invention employs some associated hardware.

Looking now at Figure 3 which shows a simplified schematic diagram representing the associated hardware employed by the present invention for active power conservation. When monitor 40 (not shown) determines the CPU is ready to sleep, it writes an I/O port (not shown) which causes a pulse on the SLEEP line. The rising edge of this pulse on the SLEEP line causes flip flop 500 to clock a high to Q and a low to Q-. This causes the AND/OR logic (AND gates 510, 520; OR gate 530) to select the pulses travelling the SLEEP CLOCK line from SLEEP CLOCK oscillator 540 to be sent to and used by the CPU CLOCK. SLEEP CLOCK oscillator 540 is a slower clock than the CPU clock used during normal CPU activity. The high coming from the Q of flip flop 500 ANDed (510) with the pulses coming from SLEEP CLOCK oscillator 540 is ORed (530) with the result of the low on the Q- of flip flop 500 ANDed (520) with the pulse generated along the HIGH SPEED CLOCK line by the HIGH SPEED CLOCK oscillator 550 to yield the CPU CLOCK. When the I/O port designates SLEEP CLOCK, the CPU CLOCK is then equal to the SLEEP CLOCK oscillator 540 value. If, on the other hand, an interrupt occurs, an interrupt- value clears flip flop 500, thereby forcing the AND/OR selector (comprising 510, 520 and 530) to choose the HIGH SPEED CLOCK value, and returns the CPU CLOCK value to the value coming from HIGH SPEED CLOCK oscillator 550. Therefore, during any power conservation operation on the CPU, the detection of any interrupt within the system will restore the CPU operation at full clock rate prior to vectoring and processing the interrupt.

It should be noted that the associated hardware needed, external to each of the CPUs for any given system, may be different based on the operating system used, whether the CPU can be stopped, etc. Nevertheless, the scope of the present invention should not be limited by possible system specific modifications needed to permit the present invention to actively conserve power in the numerous available portable computer systems. For example two actual implementations are shown in Figures 4 and 5, discussed below.

Many VSLI designs today allow for clock switching of the CPU speed. The logic to switch from a null clock or slow clock to a fast clock logic is the same as that which allows the user to change speeds by a keyboard command. The added logic of monitor 40 working with such switching logic, causes an immediate return to a fast clock upon detection

of any interrupt. This simple logic is the key to the necessary hardware support to interrupt the CPU and thereby allow the processing of the interrupt at full speed.

The method to reduce power consumption under MS-DOS employs the MS-DOS IDLE loop trap to gain access to the "do nothing" loop. The IDLE loop provides special access to application software and operating system operations that are in a state of IDLE or low activity. Careful examination is required to determine the activity level at any given point within the system. Feedback loops are used from the interrupt 21H service request to determine the activity level. The prediction of activity level is determined by interrupt 21H requests, from which the present invention thereby sets the slice periods for "sleeping" (slowing down or stopping) the CPU. An additional feature allows the user to modify the slice depending on the activity level of interrupt 21H.

Looking now at Figure 4, which depicts a schematic of an actual sleep hardware implementation for a system such as the Intel 80386 (CPU cannot have its clock stopped). Address enable bus 600 and address bus 610 provide CPU input to demultiplexer 620. The output of demultiplexer 620 is sent along SLEEPCS- and provided as input to OR gates 630,640. The other inputs to OR gates 630,640 are the I/O write control line and the I/O read control line, respectively. The outputs of these gates, in addition to NOR gate 650, are applied to D flip flop 660 to decode the port. "INTR" is the interrupt input from the I/O port (peripherals) into NOR gate 650, which causes the logic hardware to switch back to the high speed clock. The output of flip flop 660 is then fed, along with the output from OR gate 630, to tristate buffer 670 to enable it to read back what is on the port. All of the above-identified hardware is used by the read/write I/O port (peripherals) to select the power saving "Sleep" operation. The output "SLOW-" is equivalent to "SLEEP" in Figure 2, and is inputted to flip flop 680, discussed later.

The output of SLEEP CLOCK oscillator 690 is divided into two slower clocks by D flip flops 700,710. In the particular implementation shown in Figure 4, 16 MHz sleep clock oscillator 690 is divided into 4 MHz and 8MHz clocks. Jumper J1 selects which clock is to be the "SLEEP CLOCK".

In this particular implementation, high speed clock oscillator 720 is a 32 MHz oscillator, although this particular speed is not a requirement of the present invention. The 32 MHz oscillator is put in series with a resistor (for the implementation shown, 33 ohms), which is in series with two parallel capacitors (10 pF). The result of such oscillations is tied to the clocks of D flip flops 730,740.

D flip flops 680,730,740 are synchronizing flip flops; 680,730 were not shown in the simplified sleep hardware of Figure 2. These flip flops are used to ensure the clock switch occurs only on clock edge. As can be seen in Figure 4, as with flip flop 500 of Figure 2, the output of flip flop 740 either activates OR gate 750 or OR gate 760, depending upon whether the CPU is to sleep ("FASTEN-") or awaken ("SLOWEN-").

OR gates 750,760 and AND gate 770 are the functional equivalents to the AND/OR selector of Figure 2. They are responsible for selecting either the "slowclk" (slow clock, also known as SLEEP CLOCK) or high speed clock (designated as 32 MHz on the incoming line). In this implementation, the Slow clock is either 4 MHz or 8MHz, depending upon jumper J1, and the high speed clock is 32 MHz. The output of AND gate 770 (ATUCLK) establishes the rate of the CPU clock, and is the equivalent of CPU CLOCK of Figure 2.

Consider now Figure 5, which depicts a schematic of another actual sleep hardware implementation for a system such as the Intel 80286 (CPU can have its clock stopped). The Western Digital FE3600 VLSI is used for the speed switching with a special external PAL 780 to control the interrupt gating which wakes up the CPU on any interrupt. The software power conservation according to the present invention monitors the interrupt acceptance, activating the next P(i)deltaT_i interval after the interrupt.

Any interrupt request to the CPU will return the system to normal operation. An interrupt request ("INTRQ") to the CPU will cause the PAL to issue a Wake Up signal on the RESCPU line to the FE3001 (not shown) which in turn enables the CPU and the DMA clocks to bring the system back to its normal state. This is the equivalent of the "INTERRUPT-" of Figure 2. Interrupt Request is synchronized to avoid confusing the state machine so that Interrupt (INTDET) will only be detected while the cycle is active. The rising edge of RESCPU will wake up the FE 3001 which in turn releases the whole system from the Sleep Mode.

Implementation for the 386SX is different only in the external hardware and software power conservation loop. The software loop will set external hardware to switch to the high speed clock on interrupt prior to vectoring the interrupt. Once return is made to the power conservation software, the high speed clock cycle will be detected and the hardware will be reset for full clock operation.

Implementation for OS/2 uses the "do nothing" loop programmed as a THREAD running in background operation with low priority. Once the THREAD is activated, the CPU sleep, or low speed clock, operation will be activated until an interrupt occurs thereby placing the CPU back to the original clock rate.

Although interrupts have been employed to wake up the CPU in the preferred embodiment of the present invention, it should be realized that any periodic activity within the system, or applied to the system, could also be used for the same function.

Claims

1. A method for implementing a real-time power conservation mode in a computer having a central processing unit (CPU), comprising the sequential steps of:

   determining in real time the level of CPU activity by detecting periods of activity (70) and inactivity (60) of said CPU;
   deciding from said level of CPU activity whether it is available for power conservation;
   if said CPU is available for power conservation, causing a hardware selector (500,510,520,530) to reduce the clock rate below the current clock rate or stop the clock provided to said CPU (280);
   maintaining the clock stopped or at the reduced rate until an interrupt occurs (320), and
   causing said hardware selector to restore the clock rate provided to said CPU to said current clock rate in response to said interrupt (330).

2. A method according to claim 1, wherein said step of causing a hardware selector to reduce the clock rate provided to the CPU below the current clock rate or stop the clock provided to said CPU (280), further comprises the step of:

   applying a power conservation CPU command via a communication line to said hardware selector;
   said hardware selector being arranged to select a power conservation clock in response to said power conservation CPU command, and to apply pulses from said power conservation clock to said CPU thereby to place said CPU in a power conservation mode.

3. A method according to claim 1 or claim 2, wherein said step of causing said hardware selector to restore the clock rate provided to said CPU to said current clock rate, further comprises the step of:

   applying a wake CPU command via a communication line to said hardware selector;
   said hardware selector being caused to select said clock having said current clock rate in response to said wake CPU command, and to apply clock pulses from said clock having said current clock rate to said CPU thereby to wake said CPU.

4. A method according to claim 1, claim 2 or claim 3 wherein said step of determining the level of CPU activity further comprises the steps of:

   ascertaining whether said CPU is already in a power conservation mode;
   and if not, determining whether there are interrupts available to restore the clock rate provided to said CPU to said current clock rate before said CPU is put into a power conservation mode.

5. A method according to claim 1, wherein said step of causing a hardware selector to reduce the clock rate below the current clock rate or stop the clock provided to the CPU, further comprises the steps of:

   determining whether there has been an increase in CPU activity; and
   adjusting a rest interval of said CPU according to that determination (20,30).

6. A method according to claim 1, wherein said interrupt is generated by a periodic activity within the computer.

7. A method according to any one of claims 1 to 6 including selecting said current clock rate from either a first clock speed provided by a first high speed clock or a second clock speed provided by a second high speed clock.

8. A method according to claim 7, wherein said step of deciding whether said CPU is available for power conservation further comprises the steps of:

   if said CPU is available for power conservation, determining the current clock rate (260) of said CPU; and
   saving a value representative of said determined current clock rate (270).

9. A method according to claim 8, wherein the step of causing said hardware selector to restore said determined current clock rate to said CPU further comprises the steps of:

   retrieving said saved value; and

setting the clock rate of said CPU to the rate represented by the retrieved value (330).

10. A method according to any one of claims 1 to 9, wherein before the step of determining the level of CPU activity, control is accepted from the operating system of the CPU, and after the step of causing said hardware selector to restore the clock rate provided to said CPU to said current clock rate, control is returned to the operating system.

11. A method according to claim 10, wherein if the decision from determining the level of CPU activity of said CPU is that said CPU is not available for power conservation then returning control to the operating system.

12. An apparatus for real time power conservation in a computer having a central processing unit (CPU), comprising :

a first clock oscillator (550) for providing pulses at a first rate;
a second clock oscillator (540) for providing pulses at a second rate, slower than said first rate; and
a hardware selector (500, 510, 520, 530) for selecting pulses from said first oscillator or from said second oscillator, said hardware selector being arranged to apply the selected pulses as clock signals to said CPU;
the apparatus being characterized by further comprising :
a CPU activity detector (10) for determining in real time the level of CPU activity by detecting periods of activity and inactivity of said CPU and providing an indication of that level of CPU activity; and
a CPU sleep manager (20,30) adapted to receive the indication of the level of CPU activity from said CPU activity detector, and to apply a signal to said hardware selector designating which pulses said hardware selector should select to apply clock signals to said CPU, according to said received indication.

13. An apparatus according to claim 12, wherein said CPU activity detector and said CPU sleep manager reside within said CPU.

14. An apparatus according to claim 12 or claim 13, wherein said CPU sleep manager further comprises:

an adjustor (20 ,30) responsive to said indication representing the current detected CPU activity level for lengthening a CPU rest period when said current detected CPU activity level has decreased or for shortening said CPU rest period when said current detected CPU activity level
has increased;
a clock rate keeper for determining a current clock rate for said CPU, for saving a value equal to said current clock rate while said CPU is receiving clock pulses at another rate for power conservation, as well as for retrieving said saved value when said CPU is to return to normal operation; and
a pulse generator for generating a pulse for designating to said hardware selector which oscillator pulses said hardware selector should select.

Patentansprüche

1. Verfahren zum Implementieren eines Echtzeit-Leistungseinsparungsmodus in einem Computer mit einer Zentraleinheit (CPU), das die aufeinanderfolgenden Schritte umfaßt:

Bestimmen des Niveaus der CPU-Aktivität in Echtzeit durch Erkennen von Perioden der Aktivität (70) und Perioden der Inaktivität (60) der CPU;
Entscheiden aufgrund des Niveaus der CPU-Aktivität, ob sie für Leistungseinsparung verfügbar ist;
Veranlassen eines Hardware-Auswahlelementes (500, 510, 520, 530), die Taktfrequenz unter die aktuelle Taktfrequenz zu reduzieren oder den zur CPU (280) gelieferten Takt zu stoppen, wenn die CPU für Leistungseinsparung verfügbar ist;
Belassen des Taktes im gestoppten Zustand oder bei der reduzierten Frequenz bis ein Unterbrechungssignal (320) erscheint, und
Veranlassen des Hardware-Auswahlelementes, die zur CPU gelieferte Taktfrequenz als Reaktion auf das Unterbrechungssignal (330) auf die aktuelle Taktfrequenz zurückzustellen.

2. Verfahren nach Anspruch 1, bei dem der Schritt des Veranlassens eines Hardware-Auswahlelementes, die zur CPU gelieferte Taktfrequenz unter die aktuelle Taktfrequenz zu reduzieren oder das zur CPU (280) gelieferte Taktsignal zu stoppen, darüber hinaus den Schritt umfaßt:

Geben eines CPU-Leistungseinsparungsbefehls über eine Kommunikationsleitung zum Hardware-Auswahlelement;
wobei das Hardware-Auswahlelement dafür eingerichtet ist, einen Leistungseinsparungstaktgeber als Reaktion auf den CPU-Leistungseinsparungsbefehl auszuwählen und Impulse von dem Leistungseinsparungstaktgeber zur CPU zu liefern, wodurch die CPU in einen Leistungseinsparungsmodus gesetzt wird.

3. Verfahren nach Anspruch 1 oder 2, bei dem der Schritt des Veranlassens des Hardware-Auswahlelementes, die zur CPU gelieferte Taktfrequenz auf die aktuelle Taktfrequenz zurückzustellen, darüber hinaus den Schritt umfaßt:

   Geben eines CPU-Aktivierungsbefehls über eine Kommunikationsleitung zum Hardware-Auswahlelement;
   wobei das Hardware-Auswahlelement veranlaßt wird, den Taktgeber mit der aktuellen Taktfrequenz als Reaktion auf den CPU-Aktivierungsbefehl auszuwählen und Taktimpulse von dem Taktgeber mit der aktuellen Taktfrequenz zur CPU zu liefern, um dadurch die CPU zu aktivieren.

4. Verfahren nach Anspruch 1, Anspruch 2 oder Anspruch 3, bei dem der Schritt des Bestimmens des Niveaus der CPU-Aktivität darüber hinaus die Schritte umfaßt:

   Festellen, ob sich die CPU noch in einem Leistungseinsparungsmodus befindet;
   und falls das nicht der Fall ist, Bestimmen, ob Unterbrechungssignale verfügbar sind, um die zur CPU gelieferte Taktfrequenz auf die aktuelle Taktfrequenz zurückzustellen bevor die CPU in einen Leistungseinsparungsmodus gesetzt wird.

5. Verfahren nach Anspruch 1, bei dem der Schritt des Veranlassens eines Hardware-Auswahlelementes, die Taktfrequenz unter die aktuelle Taktfrequenz zu reduzieren oder den zur CPU gelieferten Takt zu stoppen, darüber hinaus die Schritte umfaßt:

   Bestimmen, ob es einen Anstieg der CPU-Aktivität gegeben hat; und
   Einstellen eines Ruheintervalls der CPU gemäß dieser Bestimmung (20, 30).

6. Verfahren nach Anspruch 1, bei dem das Unterbrechungssignal durch eine periodische Aktivität in dem Computer erzeugt wird.

7. Verfahren nach einem der Ansprüche 1 bis 6, das das Auswählen der aktuellen Taktfrequenz aus entweder einer ersten Taktgeschwindigkeit, die durch einen ersten Hochgeschwindigkeits-Taktgeber geliefert wird, oder einer zweiten Taktgeschwindigkeit, die durch einen zweiten Hochgeschwindigkeits-Taktgeber geliefert wird, beinhaltet.

8. Verfahren nach Anspruch 7, bei dem der Schritt des Entscheidens, ob die CPU für Leistungseinsparung verfügbar ist, darüber hinaus die Schritte umfaßt:

   Bestimmen der aktuellen Taktfrequenz (260) der CPU, wenn die CPU für Leistungseinsparung verfügbar ist; und
   Speichern eines Wertes, der die bestimmte aktuelle Taktfrequenz (270) repräsentiert.

9. Verfahren nach Anspruch 8, bei dem der Schritt des Veranlassens des Hardware-Auswahlelementes, die bestimmte aktuelle Taktfrequenz für die CPU zurückzustellen, darüber hinaus die Schritte umfaßt:

   Wiederauffinden des gespeicherten Wertes; und
   Zurückstellen der Taktfrequenz der CPU auf die durch den wiederaufgefundenen Wert (330) repräsentierte Frequenz.

10. Verfahren nach einem der Ansprüche 1 bis 9, bei dem vor dem Schritt des Bestimmens des Niveaus der CPU-Aktivität die Steuerung von dem Betriebssystem der CPU entgegengenommen wird und nach dem Schritt des Veranlassens des Hardware-Auswahlelementes, die zur CPU gelieferte Taktfrequenz auf die aktuelle Taktfrequenz zurückzustellen, die Steuerung zum Betriebssystem zurückgegeben wird.

11. Verfahren nach Anspruch 10, bei dem in dem Fall, daß die aus dem Bestimmen des Niveaus der CPU-Aktivität resultierende Entscheidung so ausfällt, daß die CPU für Leistungseinsparung nicht verfügbar ist, die Steuerung dann zum Betriebssystem zurückkehrt.

12. Vorrichtung zur Echtzeit-Leistungseinsparung in einem Computer mit einer Zentraleinheit (CPU), umfassend:

einen ersten Taktoszillator (550), um Impulse mit einer ersten Frequenz zu liefern;
einen zweiten Taktoszillator (540), um Impulse mit einer zweiten Frequenz zu liefern, die langsamer als die erste Frequenz ist; und
ein Hardware-Auswahlelement (500, 510, 520, 530), um Impulse von dem ersten Oszillator oder von dem zweiten Oszillator auszuwählen, wobei das Hardware-Auswahlelement dafür eingerichtet ist, die ausgewählten Impulse als Taktsignale zur CPU zu liefern; wobei die Vorrichtung dadurch gekennzeichnet ist, daß sie darüber hinaus umfaßt:
ein CPU-Aktivitätserkennungselement (10) zum Bestimmen des Niveaus der CPU-Aktivität in Echtzeit durch Erkennen von Perioden der Aktivität und Perioden der Inaktivität der CPU und Liefern einer Anzeige dieses Niveaus der CPU-Aktivität; und
ein CPU-Ruhezustandsverwaltungselement (20, 30), das dafür eingerichtet ist, die Anzeige des Niveaus der CPU-Aktivität von dem CPU-Aktivitätserkennungselement zu empfangen, und ein Signal zum Hardware-Auswahlelement zu liefern, das bestimmt, welche Impulse das Hardware-Auswahlelement auswählen soll, um Taktsignale gemäß der empfangenen Anzeige zur CPU zu liefern.

13. Vorrichtung nach Anspruch 12, bei der das CPU-Aktivitätserkennungselementelement und das CPU-Ruhezustandsverwaltungselement in der CPU liegen.

14. Vorrichtung nach Anspruch 12 oder Anspruch 13, bei der das CPU-Ruhezustandsverwaltungselement darüber hinaus umfaßt:

ein Einstellungselement (20, 30), das auf die Anzeige reagiert, die das aktuelle erkannte CPU-Aktivitätsniveau repräsentiert, um eine CPU-Ruheperiode zu verlängern, wenn das aktuelle erkannte CPU-Aktivitätsniveau abgenommen hat, oder um die CPU-Ruheperiode zu verkürzen, wenn das aktuelle erkannte CPU-Aktivitätsniveau zugenommen hat;
ein Taktfrequenzerhaltungselement zum Bestimmen einer aktuellen Taktfrequenz für die CPU, sowohl um einen der aktuellen Taktfrequenz entsprechenden Wert zu speichern, während die CPU Taktimpulse mit einer anderen Frequenz zur Leistungseinsparung empfängt, als auch um den gespeicherten Wert wiederaufzufinden, wenn die CPU in den Normalbetrieb zurückkehren soll; und
einen Impulsgenerator zum Erzeugen eines Impulses, um für das Hardware-Auswahlelement zu bestimmen, welche Oszillatorimpulse das Hardware-Auswahlelement auswählen soll.

Revendications

1. Procédé pour mettre en oeuvre un mode d'économie d'énergie en temps réel dans un ordinateur ayant une unité centrale de traitement (CPU), comprenant les étapes séquentielles suivantes :

déterminer en temps réel le niveau d'activité de CPU en détectant des périodes d'activité (70) et d'inactivité (60) dudit CPU ;
décider à partir dudit niveau d'activité de CPU si le CPU est disponible pour une économie d'énergie ;
si ladite CPU est disponible pour une économie d'énergie, amener un sélecteur en matériel (500, 510, 520, 530) à réduire la fréquence d'horloge au-dessous de la fréquence courante d'horloge ou à arrêter ladite horloge fournie à ladite CPU (280) ;
maintenir l'horloge arrêtée ou à la fréquence réduite jusqu'à ce qu'une interruption se produise (320), et amener ledit sélecteur en matériel à restaurer la fréquence d'horloge fournie à ladite CPU à ladite fréquence courante en réponse à ladite interruption (330).

2. Procédé selon la revendication 1, dans lequel ladite étape consistant à amener un sélecteur en matériel à réduire la fréquence d'horloge fournie par le CPV au-dessous de la fréquence courante d'horloge ou à arrêter ladite horloge fournie à la CPU comprend en outre les étapes suivantes :

appliquer une commande de CPU d'économie d'énergie par l'intermédiaire d'une ligne de communication audit sélecteur en matériel;
ledit sélecteur en matériel étant disposé pour sélectionner une horloge d'économie d'énergie en réponse à ladite commande de CPU d'économie d'énergie, et pour appliquer des impulsions à partir de ladite horloge

d'économie d'énergie à la CPU afin de placer ainsi ladite CPU dans un mode d'économie d'énergie.

3. Procédé selon la revendication 1 ou la revendication 2, dans lequel ladite étape consistant à amener ledit sélecteur en matériel à restaurer la fréquence d'horloge fournie à la CPU à ladite fréquence courante d'horloge comprend en outre les étapes suivantes :

   appliquer une commande de réveil de CPU par l'intermédiaire d'une ligne de communication audit sélecteur en matériel ;
   ledit sélecteur en matériel étant amené à sélectionner ladite horloge ayant ladite fréquence courante d'horloge en réponse à ladite commande d'éveil de CPU, et à appliquer des impulsions d'horloge ayant ladite fréquence courante d'horloge audit CPU afin de réveiller ainsi ladite CPU.

4. Procédé selon la revendication 1, 2 ou 3, dans lequel ladite étape consistant à déterminer le niveau d'activité de CPU comprend en outre les étapes suivantes :

   vérifier si ladite CPU est déjà dans un mode d'économie d'énergie;
   et si ce n'est pas le cas, déterminer s'il existe des interruptions disponibles pour restaurer la fréquence d'horloge fournie à ladite CPU à ladite fréquence courante d'horloge avant que ladite CPU ne soit mise dans un mode d'économie d'énergie.

5. Procédé selon la revendication 1, dans lequel ladite étape consistant à amener un sélecteur en matériel à réduire la fréquence d'horloge au-dessous de la fréquence courante d'horloge ou d'arrêter l'horloge fournie à la CPU comprend en outre les étapes suivantes :

   déterminer s'il s'est produit une augmentation d'activité de la CPU; et
   ajuster un intervalle de repos de la CPU en fonction de cette détermination (20, 30).

6. Procédé selon la revendication 1, dans lequel ladite interruption est produite par une activité périodique dans l'ordinateur.

7. Procédé selon l'une quelconque des revendications 1 à 6, comprenant la sélection de ladite fréquence courante d'horloge parmi soit une première fréquence d'horloge fournie par une première horloge à grande vitesse, soit une deuxième fréquence d'horloge fournie par une deuxième horloge à grande vitesse.

8. Procédé selon la revendication 7, dans lequel ladite étape consistant à décider si la CPU est disponible pour une économie d'énergie comprend en outre les étapes suivantes :

   si ladite CPU est disponible pour une économie d'énergie, déterminer la fréquence courante d'horloge (260) de la CPU ; et
   sauvegarder une valeur représentant ladite fréquence courante déterminée d'horloge (270).

9. Procédé selon la revendication 8, dans lequel ladite étape consistant à amener ledit sélecteur en matériel à restaurer ladite fréquence courante déterminée d'horloge à la CPU comprend en outre les étapes suivantes :

   récupérer ladite valeur sauvegardée ; et
   fixer la fréquence d'horloge de la CPU à la fréquence représentée par la valeur récupérée (330).

10. Procédé selon l'une quelconque des revendication 1 à 9, dans lequel, avant l'étape consistant à déterminer le niveau d'activité de CPU, la commande est transférée depuis le système d'exploitation de la CPU, et après l'étape consistant à amener ledit sélecteur en matériel à restaurer la fréquence d'horloge fournie audit CPU à ladite fréquence courante d'horloge, la commande est rendue au système d'exploitation.

11. Procédé selon la revendication 10, dans lequel, si la décision résultant de la détermination du niveau d'activité de la CPU est que ladite CPU n'est pas disponible pour une économie d'énergie, alors la commande est rendue au système d'exploitation.

12. Appareil pour une conservation d'énergie en temps réel dans un ordinateur ayant une unité centrale de traitement (CPU), comprenant :

un premier oscillateur d'horloge (550) pour foumir des impulsions à une première fréquence ,
un deuxième oscillateur d'horloge (540) pour foumir des impulsions à une deuxième fréquence inférieure à ladite première fréquence ;
un sélecteur en matériel (500, 510, 520, 530) pour sélectionner des impulsions dans ledit premier oscillateur ou dans ledit deuxième oscillateur, ledit sélecteur en matériel étant agencé pour appliquer les impulsions sélectionnées comme signaux d'horloge à ladite CPU ;
l'appareil étant caractérisé en ce qu'il comprend en outre :
un détecteur d'activité de CPU (10) pour déterminer en temps réel le niveau d'activité de CPU en détectant des périodes d'activité et d'inactivité de ladite CPU et en fournissant une indication de ce niveau d'activité de CPU ; et
un gestionnaire de sommeil de CPU (20, 30) conçu pour recevoir l'indication du niveau d'activité de CPU dudit détecteur d'activité de CPU, et pour appliquer un signal audit sélecteur en matériel indiquant quelles impulsions ledit sélecteur en matériel devrait sélectionner pour appliquer des signaux d'horloge à la CPU en fonction de ladite indication reçue.

13. Appareil selon la revendication 12, dans lequel ledit détecteur d'activité de CPU et ledit gestionnaire de sommeil de CPU sont situés dans ladite CPU.

14. Appareil selon la revendication 12 ou 13, dans lequel ledit gestionnaire de sommeil de CPU comprend en outre :

un dispositif d'ajustement (20, 30) répondant à ladite indication représentant le niveau courant détecté d'activité de CPU pour allonger une période de repos de la CPU quand ledit niveau courant détecté d'activité de CPU a diminué ou pour raccourcir ladite période de repos de la CPU quand ledit niveau courant détecté d'activité de CPU a augmenté ;
un dispositif de conservation de fréquence d'horloge pour déterminer une fréquence courante d'horloge pour ladite CPU, sauvegarder une valeur égale à ladite fréquence courante d'horloge pendant que la CPU reçoit des impulsions d'horloge à une autre fréquence pour une économie d'énergie, ainsi que pour récupérer ladite valeur sauvegardée quand la CPU doit revenir à un fonctionnement normal ; et
un générateur d'impulsions pour produire une impulsion afin d'indiquer audit sélecteur en matériel quelles impulsions d'oscillateur ledit sélecteur en matériel devrait sélectionner.

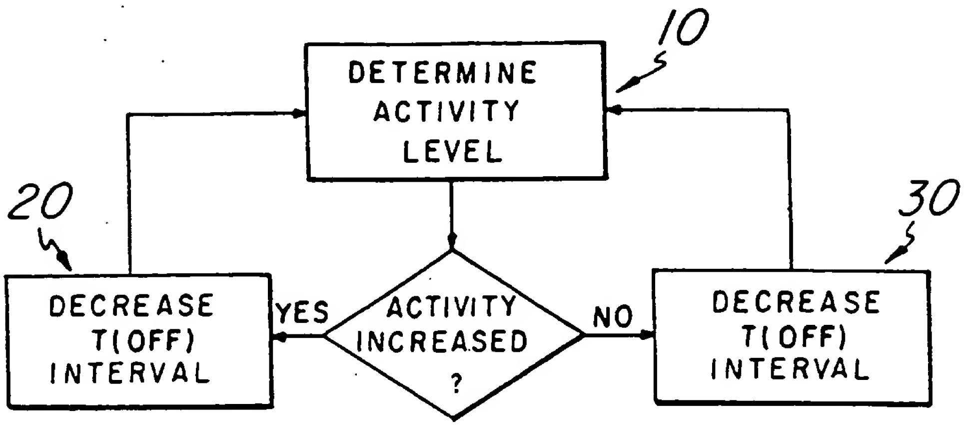
10
DETERMINE ACTIVITY LEVEL
20
DECREASE T(OFF) INTERVAL
YES
ACTIVITY INCREASED ?
NO
30
DECREASE T(OFF) INTERVAL

Fig. 1

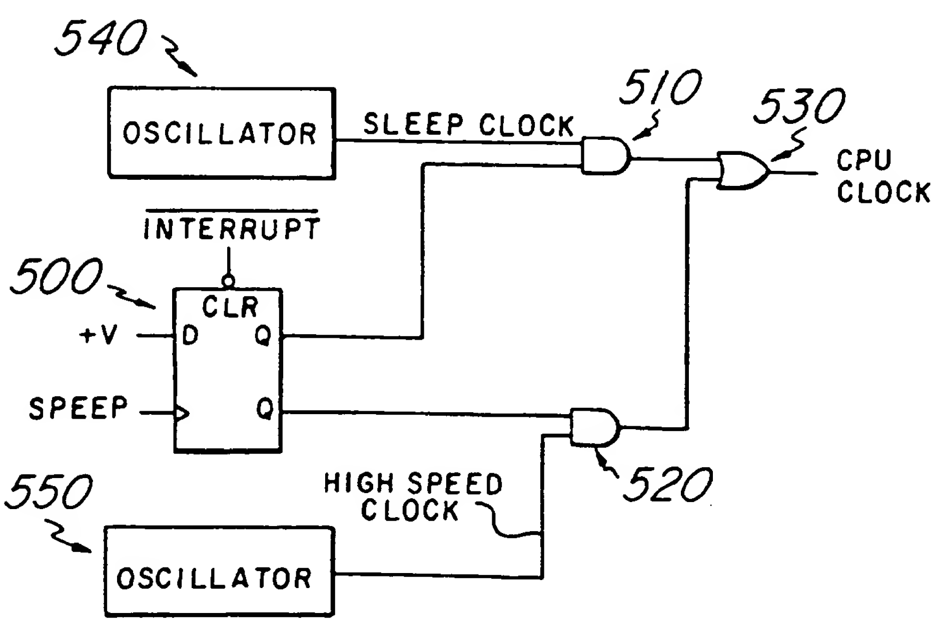
540
OSCILLATOR
SLEEP CLOCK
510
530
CPU CLOCK
INTERRUPT
500
CLR
+V
D
Q
SPEEP
Q
HIGH SPEED CLOCK
520
550
OSCILLATOR

Fig. 3

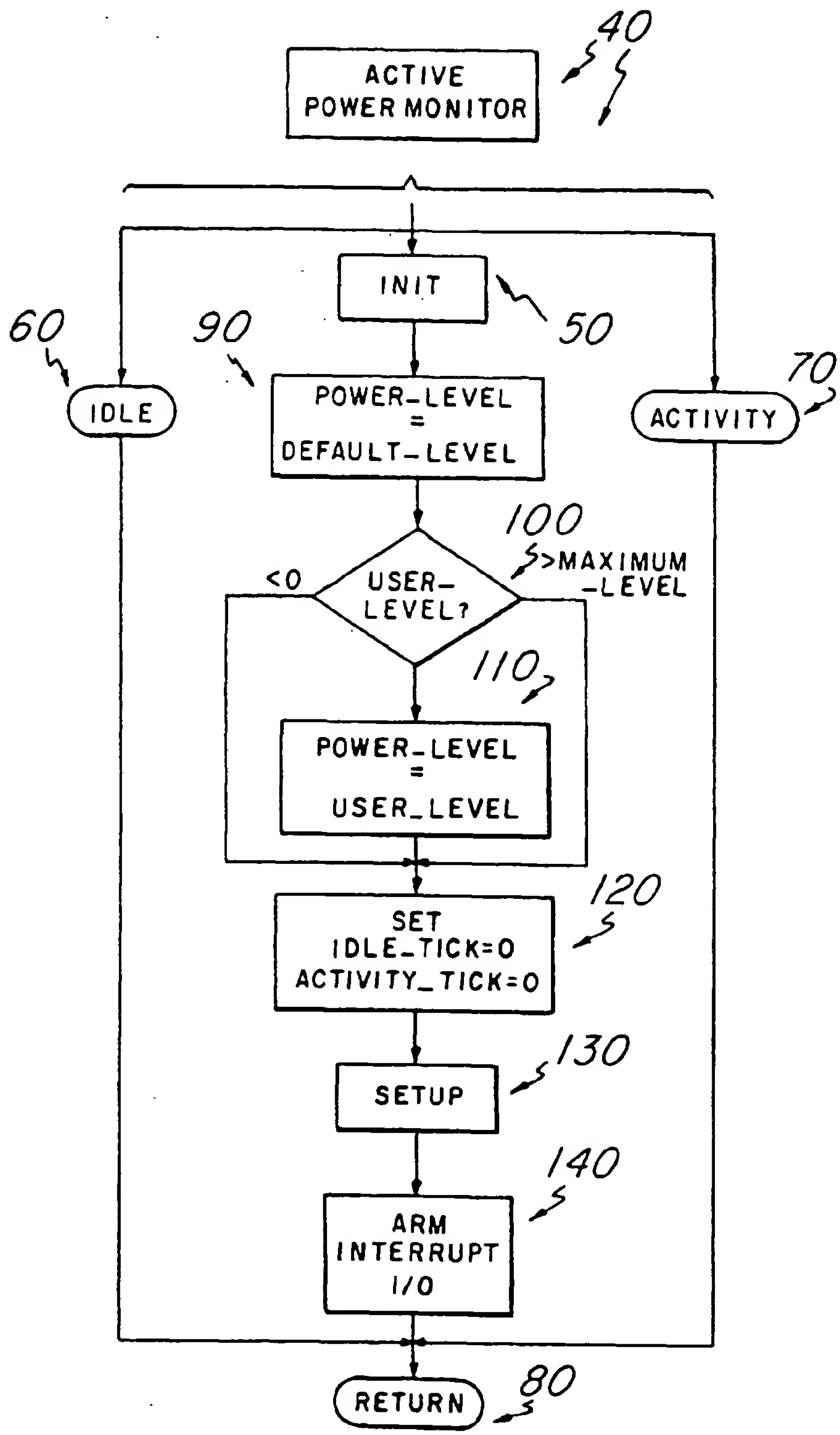
ACTIVE
POWER MONITOR
40
INIT
50
60
IDLE
90
POWER_LEVEL
=
DEFAULT_LEVEL
70
ACTIVITY
100
<0
USER_
LEVEL?
>MAXIMUM
-LEVEL
110
POWER_LEVEL
=
USER_LEVEL
120
SET
IDLE_TICK=0
ACTIVITY_TICK=0
130
SETUP
140
ARM
INTERRUPT
I/O
RETURN
80

Fig. 2a

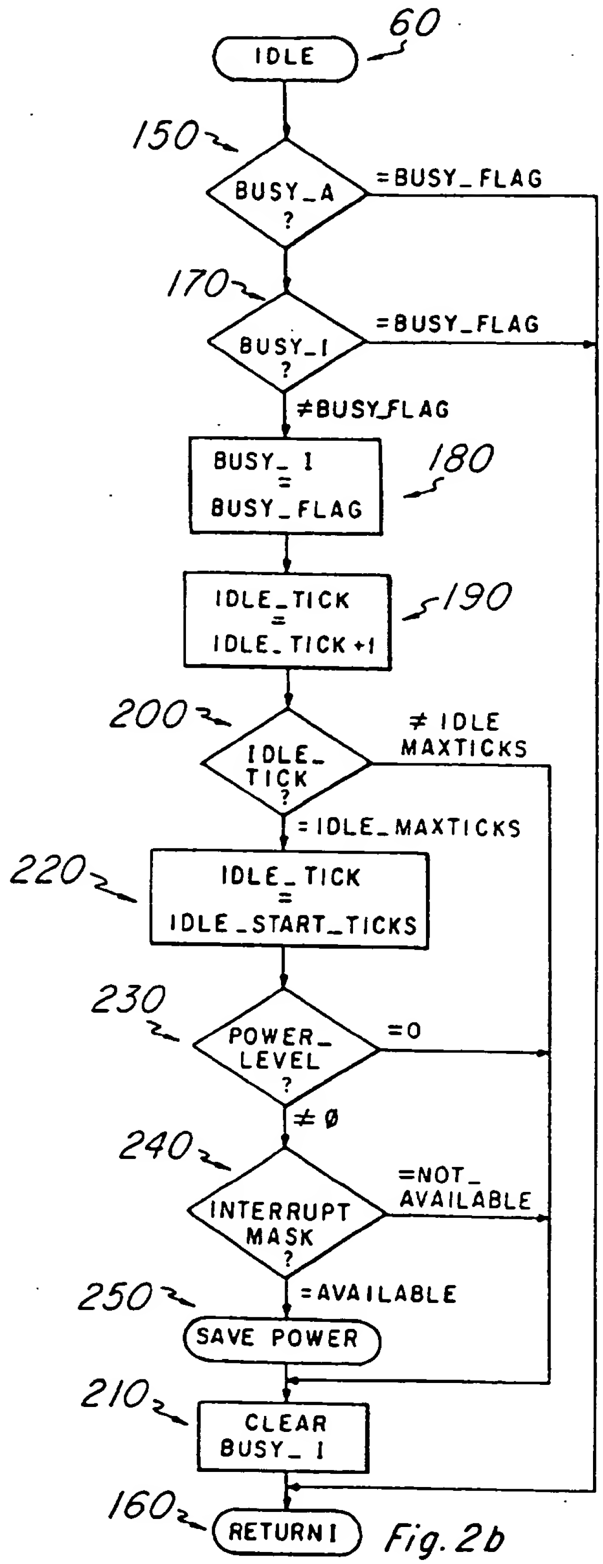
60
IDLE
150
BUSY_A ?
=BUSY_FLAG
170
BUSY_I ?
=BUSY_FLAG
≠BUSY_FLAG
BUSY_I = BUSY_FLAG
180
IDLE_TICK = IDLE_TICK +1
190
200
IDLE_TICK ?
≠ IDLE MAXTICKS
= IDLE_MAXTICKS
220
IDLE_TICK = IDLE_START_TICKS
230
POWER_LEVEL ?
=0
≠ Ø
240
INTERRUPT MASK ?
=NOT_AVAILABLE
=AVAILABLE
250
SAVE POWER
210
CLEAR BUSY_I
160
RETURN I

Fig. 2b

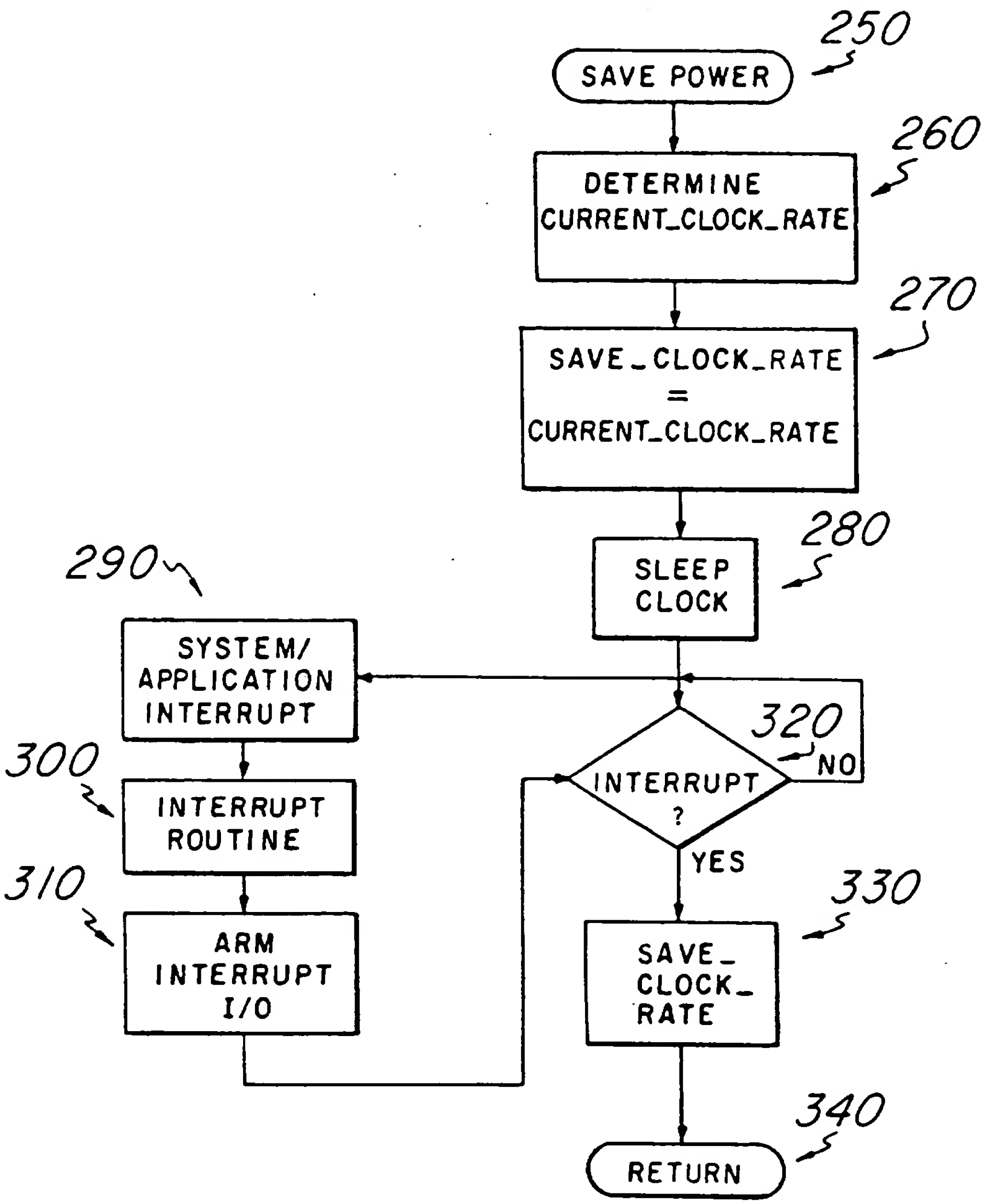
250
SAVE POWER
260
DETERMINE CURRENT_CLOCK_RATE
270
SAVE_CLOCK_RATE = CURRENT_CLOCK_RATE
280
SLEEP CLOCK
290
SYSTEM/ APPLICATION INTERRUPT
300
INTERRUPT ROUTINE
310
ARM INTERRUPT I/O
320
INTERRUPT ?
NO
YES
330
SAVE_ CLOCK_ RATE
340
RETURN

Fig. 2c

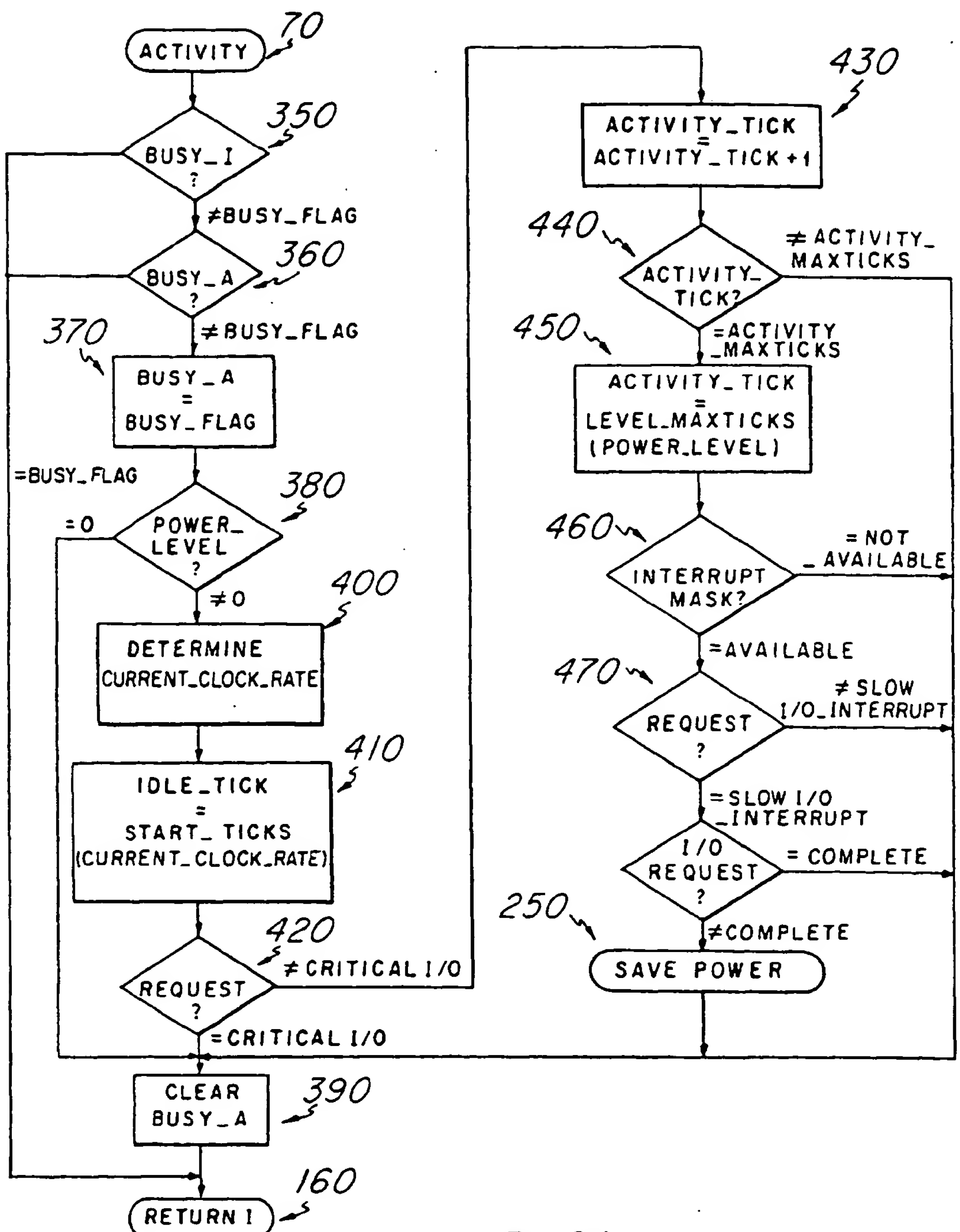
70
ACTIVITY
350
BUSY_I ?
≠BUSY_FLAG
360
BUSY_A ?
≠BUSY_FLAG
370
BUSY_A = BUSY_FLAG
=BUSY_FLAG
380
POWER_LEVEL ?
=0
≠0
400
DETERMINE CURRENT_CLOCK_RATE
410
IDLE_TICK = START_TICKS (CURRENT_CLOCK_RATE)
420
REQUEST ?
≠CRITICAL I/O
=CRITICAL I/O
390
CLEAR BUSY_A
160
RETURN I
430
ACTIVITY_TICK = ACTIVITY_TICK +1
440
ACTIVITY_TICK?
≠ACTIVITY_MAXTICKS
=ACTIVITY_MAXTICKS
450
ACTIVITY_TICK = LEVEL_MAXTICKS (POWER_LEVEL)
460
INTERRUPT MASK?
=NOT_AVAILABLE
=AVAILABLE
470
REQUEST ?
≠SLOW I/O_INTERRUPT
=SLOW I/O_INTERRUPT
I/O REQUEST ?
=COMPLETE
≠COMPLETE
250
SAVE POWER

Fig. 2d

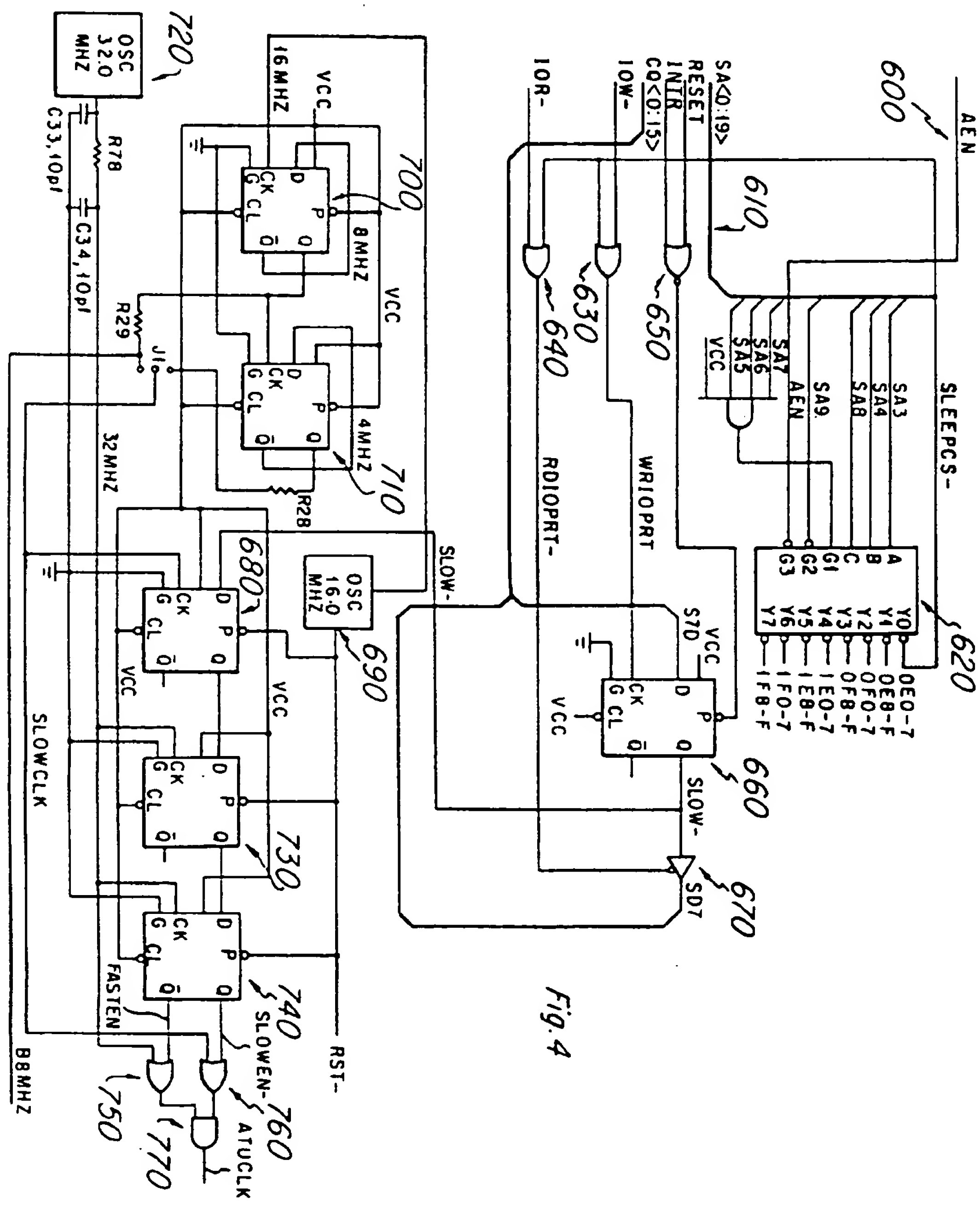
AEN
600
SLEEPCS-
SA3
SA4
SA8
SA9
AEN
SA7
SA6
SA5
VCC
SA<0:19>
RESET
INTR
CO<0:15>
IOW-
IOR-
610
620
A
B
C
G1
G2
G3
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
OEO-7
OE8-F
OFO-7
OF8-F
1EO-7
1E8-F
1FO-7
1F8-F
630
640
650
WRIOPRT
RDIOPRT-
STD
SLOW-
660
SDT
670
OSC 32.0 MHZ
720
R78
C33,10pf
C34,10pf
16MHZ
8MHZ
4MHZ
700
710
R28
R29
J1
32MHZ
SLOWCLK
OSC 16.0 MHZ
690
680
730
740
FASTEN
SLOWEN-
ATUCLK
RST-
8BMHZ
750
760
770

Fig. 4

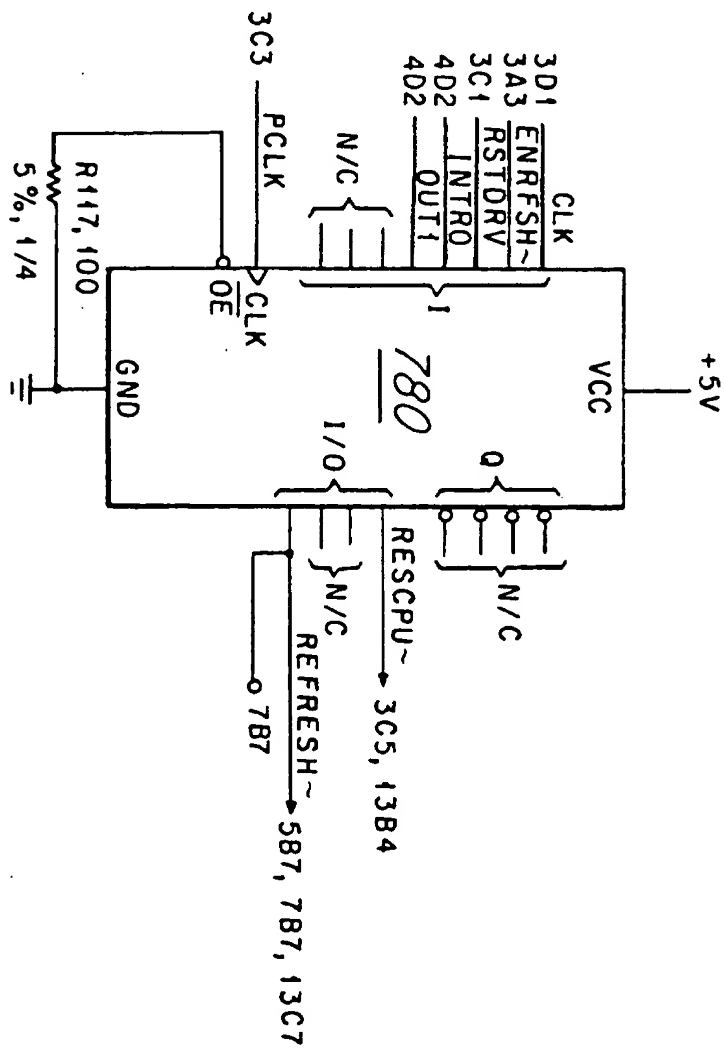
3D1
3A3
3C1
4D2
4D2
3C3
CLK
ENRFSH~
RSTDRV
INTRO
OUT1
N/C
PCLK
R117, 100
5%, 1/4
I
CLK
OE
GND
780
VCC
+5V
Q
I/O
N/C
N/C
RESCPU~
3C5, 13B4
REFRESH~
5B7, 7B7, 13C7
7B7

Fig. 5